Derivative Evaluation Function Learning Using Genetic Operators

David H. Lorenz **Shaul Markovitch**
Computer Science Department
Technion - Israel Institute of Technology
Haifa, Israel 32000
david@cs.technion.ac.il shaulm@cs.technion.ac.il

Abstract

This work studies the application of genetic algorithms to the domain of game playing, emphasising on learning a static evaluation function. Learning involves experience generation, hypothesis generation and hypothesis evaluation. Most learning systems use preclassified examples to guide the search in the hypothesis space and to evaluate current hypotheses. In game learning, it is very difficult to get classified examples. Genetic Algorithms provide an alternative approach. Competing hypotheses are evaluated by tournaments. New hypotheses are generated by genetic operators. We introduce a new framework for applying genetic algorithms to game evaluation-function learning. The evaluation function is learned by its derivatives rather than learning the function itself. We introduce a new genetic operator, called derivative crossover, that accelerates the search for static evaluation function. The operator performs cross-over on the derivatives of the chromosomes. We have demonstrated experimentally the advantage of the derivative crossover for learning an evaluation function.

Introduction

Most of the game-playing programs have a preprogrammed strategy. However, since the early days of computing, attempts have been made to build programs that learn their strategy through experience [15]. In order to address the problem of strategy learning, we first identify what exactly is to be learned, and then proceed to develop methods for carrying it out.

Game-playing learning programs traditionally assume MINIMAX search, and direct their learning efforts in acquiring a real-valued static evaluation function. Theoretically, in two-player zero-sum perfect-information games, like Checkers, one of the players possesses a winning strategy. The nodes of the game tree belong to two distinct classes: win nodes (W) and lose nodes (L). Game-learning can be therefore presented as a classification problem as following: *given a board - classify it as W or L.* Armed with such a classifier, we could play the move that transfers us to a win position. One method for achieving that is to expand the whole tree under each board, and apply the MINIMAX procedure. Since this is rarely applicable, we need a cheaper classifier that uses static features and classifies each board as win (W) or lose (L) with some level of confidence [13].

Such a classifier can be supplied by human or acquired by a program. However, it is hard to classify a board using static features even for human. Instead, human usually give an evaluation function, $V(\cdot)$, that assigns to a board a numeric value corresponding to its strength. Since at any given stage of the game, the player has only to select a move from a set of alternative moves, the value can be interpreted as: "*if* $V(j) > V(k)$ *then board* j *is more likely to be W then board* k". The assigned value acts only as a measure for comparing boards, and absolute values have no meaning by themselves [19]. Hence, we do not really need a function that orders the boards linearly. All we need is a partial order (or a preference predicate [20]) over the set of boards.

Now that we have identified *what* is to be learned, we proceed in deciding *how*. Many game learning programs rely on a supply of preclassified examples, book learning included [17]. In the absence of preclassified examples, one can use an alternative approach that lets a set of strategies compete with each other, combining successful strategies to create new ones. A natural discipline to perform learning by evolution through competition are Genetic Algorithms (GAs) [8, 7, 9]. Indeed, there have been attempts to build programs that learn to play games using GAs [4, 3, 7, 14, 9, 1, 2], but most of them deal with simple games like Tic-Tac-Toe.

In this paper we present a framework for learning the derivative function of an evaluation function rather than learning the function itself. Our framework introduces a new genetic operator, called *derivative crossover*, that accelerates the search of static evaluation function learning and makes it applicable for complex games. Each chromosome represents a classification function, and the operator performs cross-over on the derivatives of the chromosomes.

The next section discusses the view of learning as a search process. In section three we analyze the learning task and identify the representation scheme to be used by the learner. The forth section describes the derivative operators. In the fifth section we demonstrate experimentally the advantages of the derivative crossover in evaluation-function learning in the domain of Checkers. Section six concludes.

Learning As A Search Process

Learning is the process of using experience in order to modify some knowledge-base with the goal of improving it with respect to certain criteria [11]. The state of the knowledge base at any moment can be viewed as the current *hypothesis*, and the process of learning can be viewed as exploring the state space of all plausible hypotheses for the best hypothesis under the constraints imposed by experience [12]. There are three types of processes involved: *experience generation*, *hypothesis generation* and *hypothesis evaluation*. Each of these processes can be either supplied by a teacher or performed by the learner [18].

Hypotheses evaluation using sampling and competition

In the context of game learning, hypotheses are board classifiers. Classifiers are usually evaluated by the portion of classified examples they correctly classify. However, automatically producing classified boards is mostly impossible. Only relatively few boards, typically final configurations, can have their absolute classification revealed. Applying exhaustive search for producing examples is rarely feasible, and deeper partial search does not necessarily ensure a more accurate classification. Not even positions taken from a book game can be safely labeled as a win position to the actual winner of the game. The final outcome of the game can only indicate the *global performance* of the player.

We propose an alternative method of evaluating classifiers, in the context of game-playing, based on global performance alone. Since we use hypothesis evaluation to guide the search in the hypothesis space, we do not need absolute evaluation of hypotheses but rather a preference predicate to enable us select hypotheses. We define the relations "*beats*" and "*batter than*" over pairs of strategies as following:

Definition 1 *strategy A is said to* beat *strategy B, if strategy A wins more games out of n games playing against strategy B, letting $n \to \infty$.*

Definition 2 *Strategy A is said to be* better than *strategy B, if it* beats *more strategies among the set of all possible strategies.*

In order to make hypotheses evaluation practical, we *sample* both the games needed to determine the relation "*beats*", and the set of strategies needed to determine the relation "*better than*", by conducting a tournament among the set of competing strategies.

Hypotheses generation using genetic operators

Most existing programs hold one hypothesis at a time, thus performing *hill climbing* search [10]. In hill climbing, the learner begins with some initial state and steps through the space of hypotheses by modifying the current held state. Whenever required to choose between alternative states, the learner invokes an *evaluation function* for determining the most promising state. Only the new state is kept each time, and the old one is discarded. This major drawback of hill climbing often causes the search to be captured in a local extreme position. For example, Samuel's Checkers player [15, 16] initial learning attempt failed due to its use of simple hill-climbing.

It is quite possible to keep more than one alternative hypothesis. For instance, Samuel's second attempt applied hill climbing with one step look ahead and performed much better. In general the wider the search frontier, the less likely it is for the learner to be captured in a local extreme position. If we keep the set of all generated hypotheses and select the best one to be modified next, we perform *best-first* search. In best-first the entire search frontier is held in memory, expanding each time the most promising state according to the evaluation function.

In *beam search* we keep a fixed number of best nodes in memory. This has the advantage of keeping several alternative hypotheses, reducing the likelihood of being trapped in a local extreme, as well as the advantage of using restricted memory resources, making the search computationally feasible. Genetic algorithms, where a beam of models evolves through global competition, could serve as the means of generating and evaluating hypotheses. In the next two sections we first develop a method for representing hypotheses, and proceed to develop operators for traversing the hypothesis space.

```
$MINIMAX_{\mathcal{R}}(u,t,d):$

  if $(d=0) or (succ(u)=\phi)$ then return $u$;
  else case $t$ of
    $MAX$ : return $max_{\mathcal{R}}\{MINIMAX_{\mathcal{R}}(v,MIN,d-1) : v \in succ(u)\}$;
    $MIN$ : return $min_{\mathcal{R}}\{MINIMAX_{\mathcal{R}}(v,MAX,d-1) : v \in succ(u)\}$;
```

Figure 1: $Minimax_{\mathcal{R}}$ strategy

Methodology For Evaluation Function Learning

A game-playing strategy is a mapping of the board space into itself. In complex games the space of all mappings is too large to be practically traversed. We will therefore restrict the search to the space of strategies that apply MINIMAX as their control strategy. Traditionally, MINIMAX uses a real-valued evaluation function. However, MINIMAX can be easily modified to be used with any relation $\mathcal{R}$ for which $max_{\mathcal{R}}$ and $min_{\mathcal{R}}$ are defined. A $MINIMAX_{\mathcal{R}}$ procedure is shown in figure 1. The procedure returns a board rather than a value, thus $max_{\mathcal{R}}$ and $min_{\mathcal{R}}$ are well defined.

This work assumes a reduced hypothesis space consisting of all strategies that apply[1] $MINIMAX_{\mathcal{R}}$ to a limited depth. Therefore, the hypothesis space is essentially a set of relations. To ensure that $MINIMAX_{\mathcal{R}}$ operates well we require that the relations are complete, reflexive and transitive.

The representation scheme

The learner uses a representation scheme for representing hypotheses. The representation scheme is defined by a *representation space* and an *encoding function*. The encoding function maps elements of the representation space to relations in the hypothesis space. It is possible for a hypothesis to have several representatives in the representation space.

Notations: Denote by $\mathbf{B}$ the set of all plausible boards. $\mathbb{R}^{\mathbf{B}}$ denotes the set of real-valued evaluation functions $\{f\colon \mathbf{B} \to \mathbb{R}\}$. For a given set $\mathbf{I}$, $\langle \mathcal{U}, <_{\mathcal{U}} \rangle_{\mathbf{I}}$ denotes an ordered partition of $\mathbf{I}$. For each $x \in \mathbf{I}$, $[x]$ denotes a class $\mathbf{U} \in \mathcal{U}$ such that $x \in \mathbf{U}$. Let $f(\cdot)\colon \mathbf{S} \to \mathbf{T}$ be a function, and let $\mathcal{Q}$ be a relation over $\mathbf{T} \times \mathbf{T}$. $f(\cdot)\,\mathcal{Q} f(\cdot)$ denotes a relation $\mathcal{P}$ over $\mathbf{S} \times \mathbf{S}$ such that $x\mathcal{P}y \iff f(x)\,\mathcal{Q} f(x)\,,\ \forall x,y \in \mathbf{S}$.

The learning task consists of a search in the representation space, and can be defined according to the representation scheme used by the learner, in either of the following forms:

[1]Actually an Alpha-beta pruning analog algorithm was used.

1. *As a function learning problem:* The representation space is the set of real-valued functions over boards. A function $f \in \mathbb{R}^{\mathbf{B}}$ encodes the relation $f(\cdot) <_{\mathbb{R}} f(\cdot)$.
2. *As an ordering problem:* The representation space is the set of complete preorders over boards. A preorder relation $\leq_{\mathbf{B}}$ encodes itself.
3. *As a partitioning problem:* The representation space is the set of complete ordered partitions. A partition $\langle \mathcal{U}, <_{\mathcal{U}} \rangle_{\mathbf{B}}$ encodes the relation $[\cdot] <_{\mathcal{U}} [\cdot]$.
4. *As a classification problem:* The representation space is a set of functions (classifiers) that map boards into a finite ordered alphabet. A classification function $class\colon \mathbf{B} \to \mathcal{C}$, $\mathcal{C} = \langle c_1, c_2, \ldots, c_n \rangle$, $n \in \mathbb{N}$, encodes the relation $class(\cdot) <_{\mathcal{C}} class(\cdot)$.

Each of the above forms employs a different representation scheme, however, the four schemes encode the same set of relations: Given a relation $f(\cdot) <_{\mathbb{R}} f(\cdot)$, a complete preorder can be encoded as following: for all $x,y \in \mathbf{B}$ define $x \leq_{\mathbf{B}} y \iff f(x) <_{\mathbb{R}} f(y)$. Given a relation $\leq_{\mathbf{B}}$, an ordered partition $\langle \mathcal{U}, <_{\mathcal{U}} \rangle_{\mathbf{B}}$ can be encoded as following: let $\mathcal{U}$ be the set of equivalence classes of the kernel of $\leq_{\mathbf{B}}$, and define $<_{\mathcal{U}}$ according to the order between their representatives: $[x] <_{\mathcal{U}} [y] \iff x \leq_{\mathbf{B}} y$. Given a relation $[\cdot] <_{\mathcal{U}} [\cdot]$, a classification $class\colon \mathbf{B} \to \langle c_1, c_2, \ldots, c_n \rangle$ can be encoded as following: let $n = |\mathcal{U}|$ and let $(\cdot)\varphi$ be an isomorphism of $\langle \mathcal{U}, <_{\mathcal{U}} \rangle_{\mathbf{B}}$ onto $\left\langle \left\{ c_1, c_2, \ldots, c_{|\mathcal{U}|} \right\}, <_{\mathcal{C}} \right\rangle$ where the least element of $\mathcal{U}$ is mapped to c_1. Define $class_{\varphi}(x) = ([x])\varphi$ for all $x \in \mathbf{B}$. The index function $f_{class}\colon \mathbf{B} \to \mathbb{N}$ induced by the function $class_{\varphi}$ and satisfies $class_{\varphi}(x) = c_{f_{class}(x)}$, is in particular a real function, thus concluding the equivalence cycle.

Nevertheless, the effort required for searching the hypothesis space in the four representation schemes is quite different. Learning a real-valued function is the most natural thing to do, however, the representation space is the largest; different functions may induce the same preorder and thus represent the same playing strategy. We would have liked to search the space of preorders or alternatively the space of ordered partitioned, since both evade redundancy. These representation schemes, however, are difficult to code efficiently. Instead, representing a preorder by a classification function can be coded more naturally, but requires deciding on the number of classes, n, which can lead to two types of problems. Let $n_{optimal} = |\mathcal{U}|$. Either $n < n_{optimal}$ in which case the representation scheme is too restrictive, or $n > n_{optimal}$ in which case the representation scheme is too expressive. In both cases the same relation may have multiple representations. We would like to partition the space of classifiers into equivalent classes such that $class_1 \in [class_2]$ iff $class_1$ and $class_2$ represent the same preorder $\leq_{\mathbf{B}}$. Such a partition is indeed represented by a set of *canonical classification functions.*

Definition 3 *Let* $\boldsymbol{B}$ *be the set of plausible boards, and let* $\mathcal{C} = \langle c_1, c_2, \ldots, c_n \rangle$ *be an ordered set of classes. A classification function* $class\colon \boldsymbol{B} \rightarrow \mathcal{C}$ *is said to be a* canonical classification function *if class is a mapping of* $\boldsymbol{B}$ *on a proper prefix of the ordered set of classes* $\mathcal{C}$, $\{c_1, c_2, \ldots, c_m\} \subset \mathcal{C}$, $m \leq n$.

Any classification function has a performance equivalent canonical representative ($class_\varphi$ mentioned above). The space of canonical classification functions is equal in size to the space of ordered partitions. The following equation summarizes the sizes of the different representation spaces.

$$(1) \quad \begin{array}{rl} & \left|\left\{class\colon \mathbf{B} \rightarrow \langle c_1, \ldots, c_{n<n_{optimal}} \rangle\right\}\right| < \\ < & \left|\left\{\langle \mathcal{U}, <_{\mathcal{U}} \rangle_{\mathbf{B}}\right\}\right| = \\ = & \left|\left\{class\colon \mathbf{B} \rightarrow \langle c_1, \ldots, c_{n=n_{optimal}} \rangle\right\}\right| = \\ = & \left|\left\{\leq_{\mathbf{B}}\right\}\right| \leq \\ \leq & \left|\left\{class\colon \mathbf{B} \rightarrow \langle c_1, \ldots, c_{n>n_{optimal}} \rangle\right\}\right| < \\ < & |\mathbb{R}^{\mathbf{B}}| \end{array}$$

Derivative Learning

Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_L$ be L static attribute functions, $A_i\colon \mathbf{B} \rightarrow \{a_1, a_2, \ldots, a_{k_i}\}$, each mapping boards into a discrete range of values. Denote by $range(\mathcal{A}_1) \times range(\mathcal{A}_2) \times \ldots \times range(\mathcal{A}_L)$ the Cartesian vector space representing boards. We consider the evaluation function as a *two phase* mapping. $\mathbf{B} \longrightarrow range(\mathcal{A}_1) \times range(\mathcal{A}_2) \times \ldots \times range(\mathcal{A}_L) \xrightarrow{f} \langle c_1, c_2, \ldots, c_n \rangle$. A single attribute classifier is a mapping $f\colon range(\mathcal{A}) \rightarrow \mathcal{C}$.

We consider evaluation functions as a classifiers that map values of a continuous valued feature into linearly ordered classes. In order to apply genetic algorithms for searching the space of possible classifiers, we will represent each alternative classifier by a chromosome string of length l. Each chromosome is represented as a list of classes, whose i^{th} element is the class assigned to boards with feature value a_i. We use the natural vector representation as commonly used in genetic algorithms, but tend to give a relative interpretation, and insist that each strategy has a unique representation. Three performance equivalent classification functions are shown in figure 2. To prevent coding from being redundant, in

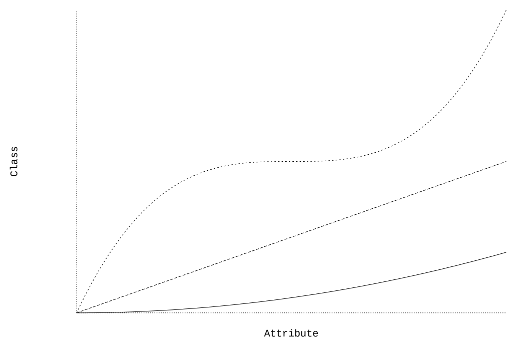

Figure 2: Performance-equal evaluation functions

the sense that the number of possible encodings exceed the number of classifiers, a canonical representation is used: all indexes are set positive letting the smallest be 1, and the entire range of classes is used.

The continuity assumption

As all the attributes considered are discrete and continues, we heuristically assume that board positions with similar attribute values have similar position strength, and thus belong to similar classes. This assumption is easily incorporated by adding a restriction forcing successive attribute values to be mapped to neighboring or identical classes. The corresponding classification function is hence assumed to be *smooth.*

Given an attribute $range(\mathcal{A}) = \langle a_1, a_2, \ldots, a_k \rangle$, define $||a_i - a_j|| \stackrel{\Delta}{=} |i - j|$, and given an ordered set of classes $\mathcal{C} = \langle c_1, c_2, \ldots, c_n \rangle$ define $||c_i - c_j|| \stackrel{\Delta}{=} |i - j|$. The desired smoothness of the evaluation function may be imposed by keeping M-bounded slopes:

$$(2) \quad \forall a_i, a_j \in range(\mathcal{A})\,, \quad \frac{||f(a_i) - f(a_j)||}{||a_i - aj||} \leq M$$

If we denote

$$(3)\; f'(a_i) = \frac{||f(a_i) - f(a_{i-1})||}{||a_i - a_{i-1}||} = \frac{||f(a_i) - f(a_{i-1})||}{1}$$

then having

$$(4) \qquad ||f'(a)|| \leq M \quad \forall a \in range(\mathcal{A})$$

guarantees M-bounded slopes.

For each M denote by $\mathcal{F}_M$ the set of M-bounded functions. Under the continuity assumption it is sufficient to search within the space of $\mathcal{F}_M$, or by following the derivative notation, within the space of functions with an M-bounded derivative.

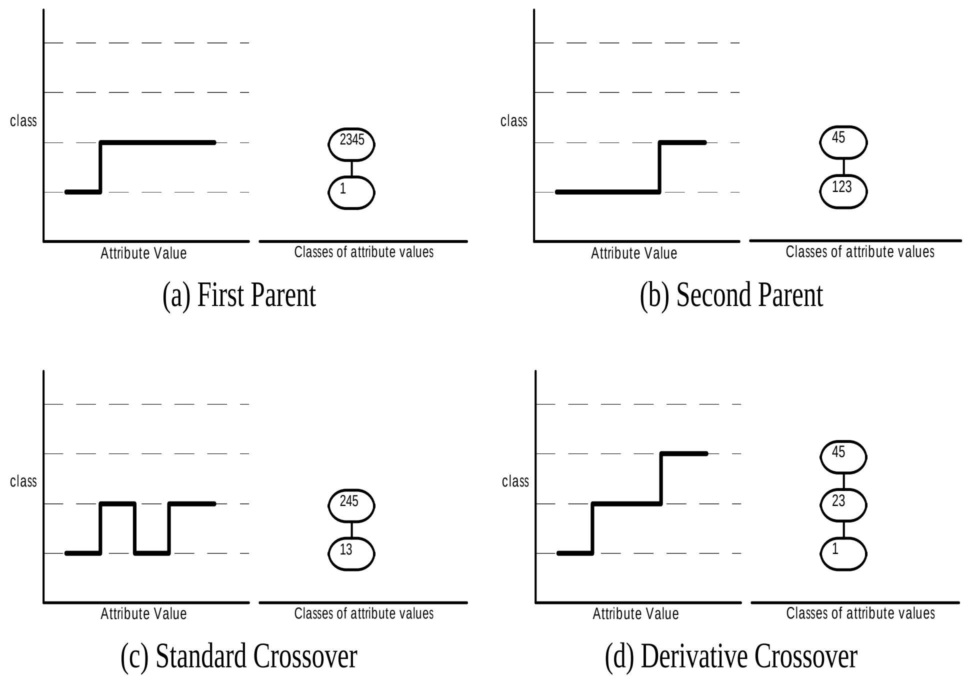

Figure 3: Motivation for using derivative crossover

Derivative Operators

The traditional crossover is one-point-crossover where two fixed length strings are cut at a random selected point, called the crossover point, and recombined by switching their parts. The conventional mutation is an unary operator where a single character at a randomly selected point, called the mutation point, is changed. We present a derivative variation of the crossover and mutation operators.

Let l be the length of the chromosomes, and let $\mathcal{C} = \langle c_1, c_2, \ldots, c_n \rangle$ be an ordered alphabet, and assume $n \geq l$. A chromosome of length l over the alphabet $\mathcal{C}$ may be viewed as a function that maps the indexes $\langle 1, 2, \ldots, l \rangle$ into the alphabet $\mathcal{C}$. The *derivative crossover* performs one-point crossover on the derivatives of the two functions, and then integrates the results to produce the new offspring. The *derivative mutation* performs conventional mutation to the function's derivative, and then integrates the result to produce the mutated offspring.

Definition 4 *Let $f = \langle t_1, t_2, \ldots, t_l \rangle$ be a vector over the given alphabet $\mathcal{C}$, and denote the derivative of f by:*

$$(5) \qquad f' = \langle ||t_2 - t_1||, ||t_3 - t_2||, \ldots, ||t_l - t_{l-1}|| \rangle$$

Let $f' = \langle d_1, d_2, \ldots, d_{l-1} \rangle$ be a vector of integers, and denote the integral of f' by:

$$(6) \qquad \int f' = \left\langle c_k, c_{k+d_1}, c_{k+d_1+d_2}, \ldots, c_{k+\sum_{i=1}^{l-1} d_i} \right\rangle$$

where $k = 1 - min\left\{0, d_1, d_1 + d_2, \ldots, \sum_{i=1}^{l-1} d_i\right\}$ is a constant keeping the indexes positive.

The derivative crossover and mutation are defined as follows:

$$(7) \qquad \Delta \mathit{Crossover}(f_1, f_2) \triangleq \int \mathit{Crossover}(f_1', f_2')$$

$$(8) \qquad \Delta \mathit{Mutation}(f) \triangleq \int \mathit{Mutation}(f')$$

Example 1 *Performing a derivative crossover of two chromosomes of length 5 over the ordered alphabet $\langle a, b, c, d, e \rangle$ may yield the following result:*

$$\begin{aligned} & \Delta \mathit{Crossover}([abcbc], [abbcd]) = \\ = \; & \textstyle\int \mathit{Crossover}([\underbrace{1, 1}_{I}, \underbrace{-1, 1}_{II}], [\underbrace{1, 0}_{III}, \underbrace{1, 1}_{IV}]) = \\ = \; & (\textstyle\int([\underbrace{1, 1}_{I}, \underbrace{1, 1}_{IV}]), \textstyle\int([\underbrace{1, 0}_{III}, \underbrace{-1, 1}_{II}])) = ([abcde], [abbab]) \end{aligned}$$

Example 2 *Similarly, performing a derivative mutation might result in:*

$$\begin{aligned} & \Delta \mathit{Mutation}([abcbc]) = \\ = \; & \textstyle\int \mathit{Mutation}([1, \underbrace{1}, -1, 1]) = \\ = \; & \textstyle\int([1, \underbrace{-1}, -1, 1] + \underbrace{2}_{k}) = [bcbab] \end{aligned}$$

Motivation for using derivative operators

Suppose we are trying to learn a strategy by using a single attribute with five values: 1,2,3,4,5. Assume that the following two promising strategies have emerged.

One which succeeded in learning the benefit of positions with attribute values 2,3,4,5 over positions with attribute value 1, and one which failed yet to distinguish between 1 and 2 but managed to learn an equally significant relative benefit of positions with attribute values 4,5 over positions with attribute values 1,2,3. Figure 3(a) shows the partition $\{\{1\},\{2,3,4,5\}\}$, and 3(b) shows the partition $\{\{1,2,3\},\{4,5\}\}$. A traditional crossover would most likely produce the undesired partition $\{\{1,3\},\{2,4,5\}\}$ shown in 3(c). On the other hand, applying the derivative crossover you most likely produce the partition {{1},{2,3},{4,5}} as shown in 3(d) combining both parents' information and automatically introducing a third new class.

Experimental Results

In order to test the framework presented in the previous sections, a series of experiments was conducted. The purpose of the experiments was to determine whether our analysis, that the derivative crossover and mutation perform better than the standard operators, would prove itself in a real game playing domain. The experiments were carried out in the following manner: the population size was kept fixed at 64, the number of generations was 40, the initial population of vectors was generated at random with the restriction of bounded derivatives. In each generation a full tournament of games was conducted using the Alpha-beta pruning algorithm with the chromosomes as the classifiers applied to the leaves. Fitness values were assigned according to overall performance. Derivative crossover was applied with probability of 0.1 and derivative mutation with decreasing probability. Rank selection was used. The experiments were conducted on a massively parallel commercial computer. A modular approach based on a parallel programming paradigm called Linda [5] was used to parallelize the experiments.

Derivative vs. absolute

In the first series of experiments the task was to learn the best evaluation function for Checkers, based on a single attribute. The attribute picked was material credit, i.e. *player's pieces − opponent's pieces*, for two reasons. First, material credit is known to be a dominating feature in Checkers and thus meaningful as a sole attribute. Secondly, the target function is known, allowing results to be assessed. Figures 4 and 5 show the best individual in selected generations, comparing derivative and standard operators. It can be observed that, for a single attribute, a stable vector was discovered after only 6 generations with derivative crossover; whereas with traditional crossover it required about 30. In both cases the middle section (-5,5) of the evolving vector resembles the expected monotonically increasing target function. However, the stable vector of the derivative case is much smoother than the absolute case vector. Using the derivative crossover achieved convergence in one fifth of the number of generations needed when using traditional one-point crossover. Since the usage of a genes falls exponentially as the location of the gene moves away from the center, many more games ought to be played each generation in order to learn the edges of the chromosome. In the derivative case there is hope for further improvement, because the edges can be independently improved without altering the established middle part. In the standard case, however, where values are absolute, all the genes need to be changed for improving the function, thus requiring more learning resources.

The benefit of derivative learning compared to learning with the standard genetic operator, is seen in figure 6. The graph shows the performance of best in-

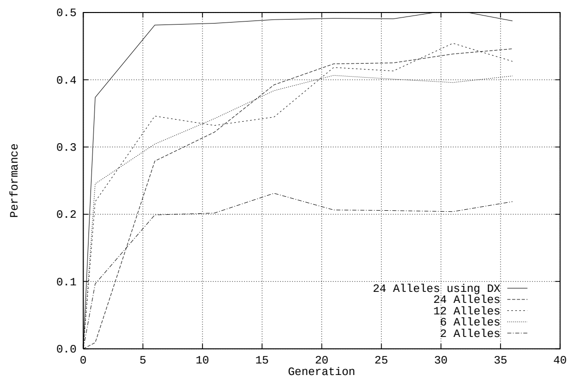

Figure 6: Derivative learning vs. direct learning

dividuals of the first 40 generation, tested against an external player. The solid line (DX) shows the derivative convergence, and the other lines show the curve of learning a classification function using standard genetic operators, with 2, 6, 12, and 24 permitted classes (alleles), respectively.

Function learning

In order to test the system capability of learning any continuous evaluation function, we have created an artificial game where the optimal evaluation function is known. To play this game, we plug in an arbitrary function, e.g. $\sin x$, and randomly generate pairs of points $\{x, y\}$, (in this experiment a set of 500 pairs). Each player orders each of the pairs. An ordered pair $\langle x, y\rangle$ is considered a correct answer if $\sin x < \sin y$. The player that has more correct answers wins the game, without being told which of the answers were correct. This simulates game experience based on overall performance using a preference predicate $\sin(\cdot) < \sin(\cdot)$. Similar learning sessions were conducted with the func-

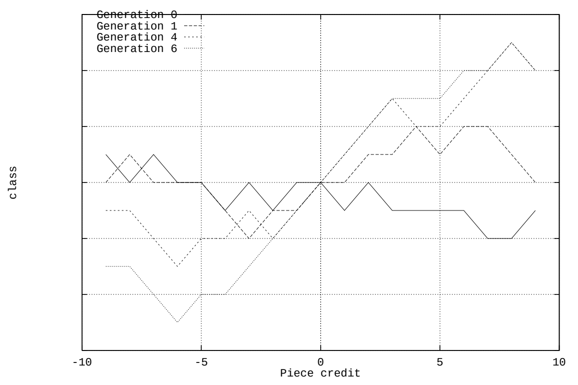

Figure 4: Chromosomes during learning using derivative operators

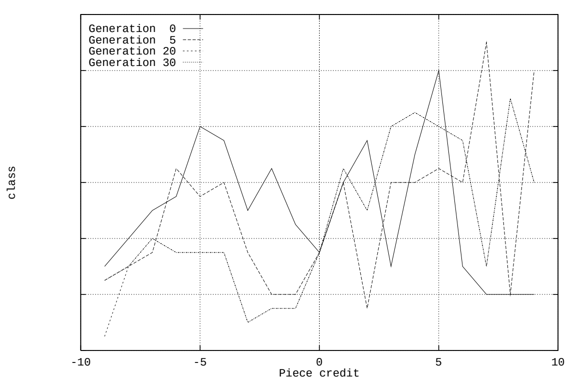

Figure 5: Chromosomes during learning using standard operators

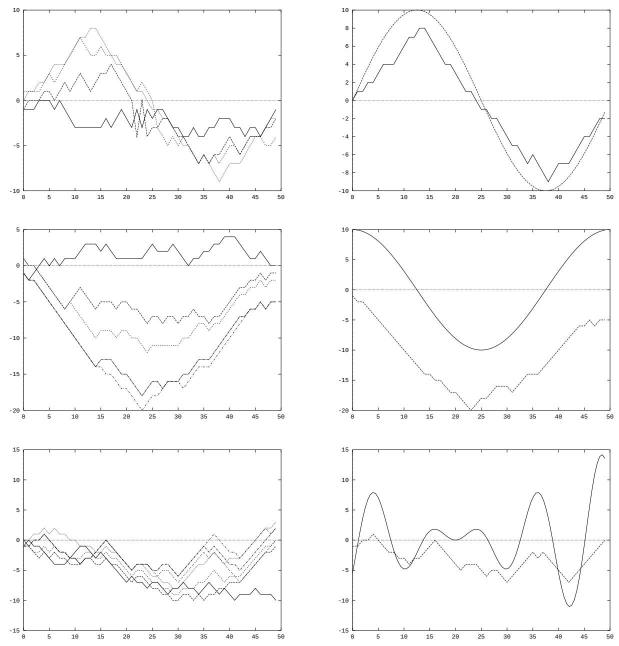

Figure 7: Best chromosome in three learning sessions compared with target function $f(x) = 10 sin \frac{2\pi}{50}x$, $f(x) = 10\cos\frac{2\pi}{50}x$, $f(x) = \frac{x-20}{2}\sin\frac{x-20}{2}$, respectively.

tions $\cos x$ and $x \sin x$. Figure 7 shows selected chromosomes during learning (left), and the final chromosome compared with the target function (right).

Combining several attributes

Combining several attributes is not a straightforward extension. Just as the final outcome of a game can only indicate the global performance of the player, the final outcome of the evaluation function indicates its global strength, and not the quality of its components separately. One approach for dealing with this difficulty if to use matrices instead of vectors. There are matrix genetic operators [6], but generalizing them to n-dimensions is not practical. An alternative approach is to learn the second attribute function on top of the first, e.g. a delta-coding genetic algorithm [21].

We learn the second attribute function in a manner similar to the first attribute, except that performance is evaluated using a linear combination of the two functions. The learning is repeated in search of the best combining coefficient, keeping the first attribute function constant. Experiments for combining *piece advantage* and *king advantage* for the game of Checkers resulted in the expected ratio of 2 : 3 combination. Combining up to 5 attributes resulted in an improvement of play. These results are shown in figure 8 .

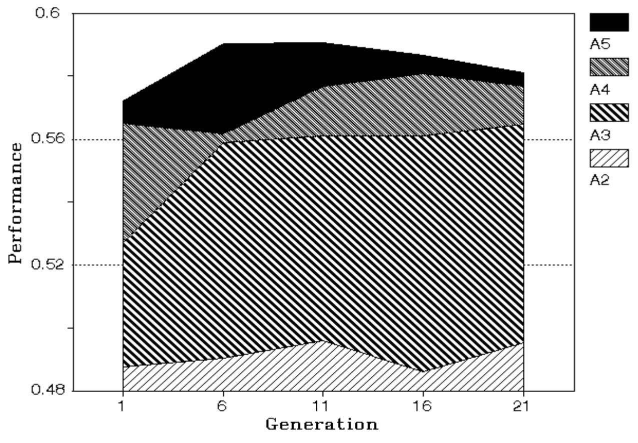

Figure 8: Performance of learners against external tester. Combinations of 2, 3, 4 and 5 attributes were used.

Interleaving derivative learning with linear combination is much more powerful than simple linear combination of features. Samuel's evaluation function was restricted to hyperplanes, a fact considered to be a major drawback of his program. Our method, however, allows surfaces such as $t_1^2 - t_2^2$ shown in figure 9 to be learned, that could not had been learned using simple linear combination of features.

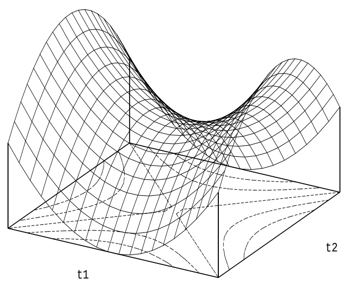

Figure 9: Representable non linear surface

Conclusions

Genetic algorithm is a natural candidate for learning in games. In this paper we study the use of genetic algorithms in game-playing learning programs, emphasising on learning a static evaluation function. Instead of blindly fitting the problem to genetic algorithm, we modified genetic algorithms to fit the problem, achieving better results. We began by deciding to invest our efforts in learning an evaluation function (rather than learn a complete strategy from scratch). We further determined that the precise learning task is finding an order relation (rather than absolute classifications). We then concluded that it is beneficial to learn an evaluation function by its derivatives, and designed a *derivative crossover* operator that combines good building blocks. A heuristic continuity assumption has proven advantageous in the genetic search. In a series of experiments performed with the new framework, the program acquired an evaluation function component for the game of Checkers, an arbitrary continuous function, and a non linear combination of features.

Previous work that use genetic algorithms to learn, either apply too demanding techniques that are not applicable for complex games, or perform a very restricted type of learning. We assumed the control to be known a priori, and left the learner with the task of learning the evaluation function. This is different than Koza's approach [9] that induces Lisp programs from scratch. There is another difference in approaches between other systems and ours concerning the coding policy. We claim that relative representation is more beneficial than absolute representation, for achieving generalization using genetic algorithms. The 1-bounded slops assumption may be generalized to M-bounded slops, increasing the expressive power of the representation, paying in search time.

The derivative operators were designed for evolving game-playing evaluation functions, but may be useful for other order-preserving representations as well. We hope that the methodology of learning the derivatives instead of the function itself can be applied with methods other than genetic algorithms, and that the derivative crossover can be applied to other areas of genetic search beyond game playing.

Acknowledgments

We want to thank Aythan Avior, Yuval Sherman, Yaron Sella, Dudi Carmel, Larry Rendell, and Joseph Gil, for contributions in different phases of the research.

References

[1] P. J. Angeline and J. B. Pollack. Coevolving hight-level representations. LAIR 92-pa-coevolve, The Ohio State University, Columbus, Ohio 43210, 1992. Submitted to the Proceedings of Artificial Life III.

[2] P. J. Angeline and J. B. Pollack. Learning with a competitive population. LAIR 92-pa-compete, The Ohio State University, Columbus, Ohio 43210, 1993. Submitted to IJCAI '93.

[3] R. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In D. Lawrence, editor, *Genetic Algorithms and Simulated Annealing*, chapter 3, pages 30–41. Pittman, 1987.

[4] J. D. Bagley. *The behavior of adaptive systems which employ genetic and correlation algorithms.* PhD thesis, University of Michigan, 1967.

[5] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, Apr 1989.

[6] B. R. Fox and M. B. McMahon. Genetic operators for sequencing problems. In G. J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 284–300. Morgan Kaufmann, 1991.

[7] D. E. Goldberg. *Genetic Algorithms in Search Optimization & Machine Learning.* Addison:Wesley, 1989.

[8] J. H. Holland. *Adaptation in Natural and Artificail Systems.* The MIT Press, 1992.

[9] J. R. Koza. *Genetic Programming - on the programming of computers by means of natural selection.* The MIT Press, 1992.

[10] P. Langley, J. H. Gennari, and W. Iba. Hill-climbing theories of learning. In *Proceedings of the 4^{th} International Workshop on Machine Learning*, number 4, pages 312–323, 1987.

[11] S. Markovitch and P. D. Scott. Information filtering: Selection mechanisms in learning systems. *Machine Learning*, 10:113–151, 1993.

[12] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203 – 226, 1982.

[13] J. R. Quinlan. Learning efficient classification procedures and their application to chess and games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 15, pages 463–482. Tioga, 1983.

[14] L. Rendell. Induction as optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):326–338, Mar 1990.

[15] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3(3):211–229, July 1959.

[16] A. L. Samuel. Some studies in machine learning using the game of checkers.II - recent progress. *IBM Journal*, November 1967.

[17] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. Reviving the game of checkers. In D. Levy and D. Beal, editors, *Heuristic Programing in Artifical Intelligence; The Second Computer Olimpiad*, pages 119–136. Ellis Horwood, London, 1991.

[18] V. L. Shalin, E. J. Wisniewski, K. R. Levi, and P. D. Scott. A formal analysis of machine learning systems for knowledge acquisition. In *Machine Learning and Uncertain Reasoning*, volume 3 of *Knowledgebased systems*, pages 75–92. 1990.

[19] P. E. Utgoff and J. A. Clouse. Two kinds of training information for evaluation function learning. In *Proc. of AAAI-91*, pages 596–600, 1991.

[20] P. E. Utgoff and S. Saxena. Learning a preference predicate. In *Proceedings of the 4^{th} International Workshop on Machine Learning*, pages 115–121. Morgan Kaufman, 1987.

[21] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In *Proc. of the Fourth International Conference on Genetic Algorithms*, pages 77–85, San Diego, CA, 1991.